NASA Technical Memorandum 88263

An Engineering Approach to the Use of Expert Systems Technology in Avionics Applications

Eugene L. Duke, Victoria A. Regenie, Marylouise Brazee, and Randal W. Brumbaugh

May 1986

NASA
National Aeronautics and Space Administration

NASA Technical Memorandum 88263

An Engineering Approach to the Use of Expert Systems Technology in Avionics Applications

Eugene L. Duke, Victoria A. Regenie, and Marylouise Brazee
Ames Research Center, Dryden Flight Research Facility, Edwards, California
Randal W. Brumbaugh
PRC Kentron, Edwards, California

1986

National Aeronautics and
Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California 93523

AN ENGINEERING APPROACH TO THE USE OF EXPERT SYSTEMS TECHNOLOGY IN AVIONICS APPLICATIONS

Eugene L. Duke, Victoria A. Regenie, and Marylouise Brazee
NASA Ames Research Center
Dryden Flight Research Facility
Edwards, California

Randal W. Brumbaugh
PRC Kentron
Edwards, California

Abstract

This paper presents the concept of using a *knowledge compiler* to transform the knowledge base and inference mechanism of an expert system into a conventional program. The motivation for this discussion is the need to accommodate real-time systems requirements in applications such as embedded avionics. The paper presents an overview of expert systems and a brief comparison of expert systems and conventional programs. Avionics applications of expert systems are briefly discussed before the detailed discussions of applying the proposed concept to example systems using forward- and backward-chaining.

Introduction

Expert systems technology offers tremendous potential for the next generation of avionics systems. The power of this technology lies both in its separation of domain specific knowledge from program control mechanisms and in the development methodology and development environment. However, as expert systems technology moves out of the research laboratory and into severely demanding applications such as avionics, new problems arise. While the power of expert systems is evident in the laboratory environment, many of the most attractive features of this technology become burdensome in applications. It is argued in this paper that, by treating expert systems as development tools for more conventional programs, many of the problems emerging in applications may be solved. It is the thesis of this paper that an engineering approach to the use of expert systems technology in avionics application may minimize the need for special purpose computers.

The concept of using an expert system as a development tool for a conventional program arose from the applications research at the Dryden Flight Research Facility of the NASA Ames Research Center (Ames-Dryden) and was suggested by the research at the University of California, Los Angeles (UCLA), under the direction of Professor Jacques Vidal. This research in real-time mechanizations of expert systems has influenced some of the key ideas presented in this paper. In particular, the doctoral thesis of John Helly [1] has provided a unique and original view of the transformation of a knowledge base into an equivalent logic representation.

At Ames-Dryden, expert systems technology is being applied to avionics systems on high-performance aircraft in two projects: the expert system flight status monitor for the X-29 forward-swept-wing aircraft [2],[3] and in the joint NASA/DARPA automated wingman program. Both NASA projects focus on the use of expert systems technology in real-time applications. The application areas in which this research is being conducted are such that extreme computational demands are placed on the host computer system. The complexity of high-performance aircraft place severe demands on expert systems technology. The following sections present an overview of expert systems and a comparison of expert systems and conventional programs as background before proceeding with the main argument of the paper: expert systems can be converted automatically into conventional programs and the process from development to deployment retains the best features of both types of systems.

It is the thesis of this paper that the knowledge base and inference mechanism of an expert system can be converted into a conventional program using a *knowledge compiler*. The concept of a knowledge compiler is explained using example forward- and backward-chaining expert systems. The main benefit of converting the expert system into a conventional program is increased execution speed and, hence, reduced processor requirements. The knowledge compilation method described in this paper is completely compatible with the usual environment available for expert systems development and allows the system developer to exploit the desirable features of expert systems while developing conventional programs.

Overview of Expert Systems

Artificial intelligence (AI) is described in Ref. 4:

"...the part of computer science concerned with designing intelligent computer systems,

that is, systems that exhibit the characteristics we associate with intelligence in human behavior...."

AI research focuses on understanding the basic processes of intelligence as well as on computer-based methodologies for solving difficult problems that would otherwise require human intelligence. The field of AI is concerned with a wide range of problem classes that are associated with intelligence in humans: problem solving, reasoning, understanding language, learning, robotics, automatic programming, and vision.

The research in problem solving and reasoning has led to the development of the subfield of applied AI known as expert systems in which general-purpose reasoning engines (inference mechanisms) are utilized to reason about domain specific knowledge in a target application area. An expert system is described in Ref. 5:

"An expert system is one that has expert rules and avoids blind search, performs well, reasons by manipulating symbols, grasps fundamental domain principles, and has complete weaker reasoning methods to fall back on when expert rules fail and to use in producing explanations. It deals with difficult problems in a complex domain, can take a problem description in lay terms and convert it to an internal representation appropriate for processing with its expert rules, and it can reason about its own knowledge (or lack thereof), especially to reconstruct inference paths rationally for explanation and self-justification."

This description of an expert system is more of a future goal than a current reality. The incorporation of fundamental domain principles into what are known as "deep" expert systems is at best a research topic; most expert systems have an extremely limited and shallow knowledge base. This lack of knowledge of fundamental principles contributes to the inability of current generation expert systems to reason about their knowledge in general and about the limitations of their knowledge in particular. However, many of the features listed in this description do exist in what might be termed the state of the art in application systems.

Figure 1 shows a common (although highly simplified) representation of the basic expert systems architecture. In this representation, the main structural features of an expert system are illustrated: knowledge acquisition facility, knowledge base, inference mechanism, and input-output system. The knowledge acquisition facility is the main interface between the expert system and the expert. This facility provides a mechanism for developing a knowledge base. The knowledge base consists of domain specific knowledge, generally in the form of conditional (if-then) rules. The inference mechanism reasons about specific facts using this knowledge base. In this simplified representation of an expert system, the inference mechanism would also provide explanations of conclusions reached and rules used to reach those conclusions. The knowledge acquisition facility, inference mechanism, and input-output system are part of the expert system program. After the specific facts that are to be reasoned about are provided, the knowledge base is treated as a data source that is used to direct the inference process.

Perhaps the most significant feature of current expert systems is the use of knowledge in the form of expert rules that represent domain principles. Encoding the knowledge and problem solving techniques of a domain expert into rules provides the real power of expert systems. These rules are used by an inference mechanism both to reason about specific facts in a given situation and to provide explanations of the deductions of the expert system to its user. Figure 2 shows what might be typical rules used in avionics applications. These rules might be used in flight control systems, guidance systems, and even more powerful integrated systems such as those being developed for the pilot's associate program. Figure 3 shows how these same rules would be coded in a higher order language (in this case, FORTRAN). As can be seen by comparing Figs. 2 and 3, the representation of knowledge is much clearer and more easily verified in the rules (Fig. 2) than in the FORTRAN code (Fig. 3).

Expert Systems and Conventional Programs

When comparing an expert system with a conventional program, differences in knowledge representation, control structure, and operational processes are most obvious. Expert systems use symbolic representations of knowledge and symbolic inference; conventional programs use numeric and logical representations. An expert system might be said to execute a compilation of knowledge while a conventional program might be characterized as a compilation of procedures. The structure of an expert system in which the knowledge base and inference mechanism are separated, differs from a conventional program in which the knowledge base and inference mechanism are essentially combined in the program code. The main operational differences arise from the ability of an expert system to offer explanations of its inference process and to have that inference process modified by the addition, deletion, or modification of rules. In a conventional program, explanation features are lacking and the modification of the reasoning process involves rewriting the code. Obviously, the features that characterize an expert system are intertwined and cannot be separated.

In addition to the representational, structural, and operational differences, expert systems also differ from conventional programs in their development. In a conventional program, the possible lines of reasoning must be mapped out ahead of time and then rigidly encoded into the program structure. In an expert system,

because the lines of reasoning are embodied in the knowledge base, the system can be developed incrementally. The environments in which these two types of programs are developed also differ radically. Conventional programs are most often developed in the target programming language with few support tools beyond an editor and, perhaps, a debugger. The environment for expert systems development is fundamentally far richer than that for conventional programs. Figure 4 shows the components of an expert system and illustrates the benefit of using an expert systems shell in the development process. Expert systems shells are available with inference mechanisms, a knowledge acquisition facility, and explanation capabilities already developed and in place. The knowledge engineer need only compile the domain specific knowledge of an expert and enter it into the shell. By using an expert systems shell, an expert system can be developed rapidly. Entering only a few key rules into a shell allows a system to be prototyped quickly and allows early feasibility demonstration.

It should be noted at this point that, when discussing expert systems, some distinction is occasionally made between *development* and *delivery* systems. The distinction is not entirely clear. A delivery system must at least have a knowledge base, an inference mechanism, and an explanation facility. A delivery system is probably not one with the capability for knowledge base modification, while the capability for knowledge base modification is an essential feature of a development system. This distinction between development and delivery systems will be avoided in this discussion; it will be assumed that all expert systems are capable of knowledge base modification.

Avionics Applications of Expert Systems

The main interest in using expert systems technology in avionics arises from the complexity of the missions to be performed and the demanding environments in which these missions must be performed. Consideration of these factors have forced requirements for increasingly more complex systems. The emerging generation of tactical vehicles (fighters and attack helicopters) are near the limit of what can be accomplished using a single pilot and conventional technology. Additionally, these systems require the pilot to perform at a level that is near (if not beyond) the limits of human performance. When all systems are operational, the basic tasks facing the pilot are data interpretation and subsystems integration. When problems arise in the avionics subsystems of these emerging systems, the pilot may be unable to cope with the situation because of the complexity of the subsystems and their interactions within the vehicle system as a whole. Expert systems technology offers the possibility of solving these problems by providing the pilot with useful *knowledge* and *assistance*. The main interest in using expert systems technology in avionics applications is that it may be the only means of providing the next level of systems integration.

Many avionics applications of expert systems technology are possible. These applications include examples of most of the class of expert systems problems described in Ref. 5: interpretation, prediction, diagnosis, planning, monitoring, debugging, repair, and control. Most of these applications will be embedded in other systems or will serve to integrate subsystems. Sensor fusion is an excellent example of interpretation in which multiple sensors provide diverse pieces of data that must be integrated into a coherent picture of the world outside the aircraft. The prediction problem is exemplified by tactics prediction in which the future tactics of an opponent would be predicted from past performance. Diagnosis could be applied to the isolation of a failure within a flight control system. Planning systems have obvious and immediate applications in route planning and target allocation problems. Monitoring systems could be used to assess the health and status of any of the various subsystems within the aircraft or of the effectiveness of the aircraft as a weapons system. Expert systems that diagnose, debug, or repair could be employed to provide the expertise needed in a reconfigurable control system. In this application, problems in a flight control system could be corrected by reconfiguration (repair) after specific malfunctions determined from sensor and aircraft behavior (diagnosis) had been analyzed and corrective measures had been identified (debugging). Finally, control expert systems could be used to integrate the subfunctions of an avionics system at the mission level. Obviously, this list of applications is not exhaustive, and the problem areas are not disjoint. However, this list of applications provides some insights into the problems associated with the use of expert systems in avionics applications.

All avionics software must execute in a real-time environment in which the timeframe is determined by the application; the slowest of these applications are those that interface with the pilot or involve long-term planning, and the fastest are those that involve flight control systems or weapon sensors. The need for real-time operation makes speed of execution a critical consideration in any examination of the tradeoffs between two competing pieces of software. Even with development of newer and more powerful avionics computers, software efficiency will continue to be an issue. More capable computers will merely result in greater demands; more computational power will simply result in greater expectations. Avionics computers will continue to be utilized to the maximum that can be obtained from them. The problems of real-time programming will continue to be a concern, and it is here that expert systems become burdensome.

One of the first rules of real-time programming is to develop an efficient code. As illus-

trated in the Transforming Expert Systems Into Conventional Programs section, expert systems are iterative in nature and the time required to converge to an answer varies with the situation being analyzed. While expert systems are more efficiently developed than conventional programs, expert systems are necessarily more inefficient in execution. Further, this inefficiency has nothing to do with the speed at which a computer executes LISP, FORTRAN, or any other computer language. The inefficiency of expert systems is inherent in the separation of the knowledge base and the control mechanism and in the iterative nature of the control process. Because the inefficiency of an expert system increases rapidly as the size of the knowledge base increases, some consideration has been given to dividing knowledge bases into smaller partitions. However, this latter approach does not eliminate the problem of inefficiency; it merely alleviates it.

Although some of the previously discussed applications require a direct interface with the pilot, not all do. In fact, many of these applications such as sensor fusion and flight control system reconfiguration will be imbedded in other systems and isolated from the pilot. The value of the high-level user interface characteristic of expert systems is thus unnecessary in many avionics applications. However, even if it is assumed that an explanation feature is required of the man-machine interface, the capability of knowledge base or program control modification is almost certainly not a requirement of an avionics system, whether it is an expert system or a conventional system. In fact, if any program control modifications are to be permitted in avionics systems, these will almost certainly be known, well-defined options that are built into the system.

The least consideration in avionics systems should be abandoning standard systems qualification procedures for the lure of exotic technology that allows a pilot to modify a system in real time. The requirement for an explanation in an avionics application is almost certainly weaker than the requirement for an explanation in a development system. At most, it might be expected that a first-level explanation would be required. A first-level explanation is an explanation of the last rule used to reach a conclusion. This type of explanation is in contrast to the sort of detailed backward justification available in some current expert systems.

Transforming Expert Systems Into Conventional Programs

This section describes a mechanism for converting expert systems into conventional programs. The requirements outlined in the previous section are taken as the requirements for the conversion process described here. While the examples that follow are only for simple forward- and backward-chaining inference mechanisms, these inference mechanisms can be and are being utilized for a wide variety of applications in avionics.

To illustrate how expert systems can be mapped into conventional programs, two examples of production rule systems are given: one with a forward-chaining inference mechanism and one with a backward-chaining inference mechanism. These examples, while extremely simple, will facilitate an understanding of the proposed approach. Further, these examples will serve to illustrate the tradeoffs involved in converting an expert system into a conventional program. The first two subsections provide a brief introduction to the inference mechanism, present a sample set of rules, and then show the equivalent FORTRAN representation. In the final subsection, expert systems and their transformations into conventional programs are discussed.

In the following examples, FORTRAN is used to illustrate the use of a higher order language. FORTRAN was chosen for two reasons: because of the authors' familiarity with the language and because FORTRAN is the primary computer language used in conventional scientific and engineering applications. This language is also supported on most machines by fast, efficient optimizing compilers. The perception that FORTRAN is probably unsuitable for AI applications also influenced the decision. Although at least one expert systems shell has been implemented in FORTRAN (the TIMM expert systems generation tool by General Research Corporation), FORTRAN is generally considered the antithesis of a suitable AI language. All examples of FORTRAN code presented in this paper could as easily have been shown in LISP, Ada, C, or any other higher order language.

Forward-Chaining Example

Forward-chaining is often referred to as "data-driven" inference because the rules are applied to the established facts to reach whatever conclusions are consistent with the given facts and the rules. Forward-chaining inference stops when a pass (iteration or cycle) through the rules yields no new facts and the inference process is complete. A set of example rules in which clauses have been replaced by symbols is as follows:

Rule number	Rule
1	If c *or* f then d
2	If d then e
3	If b *and* a then c
4	If a then b

Figure 5 shows the results of applying these rules with a forward-chaining inference mechanism in a situation in which only the fact a is established initially. In this simple example, five inference cycles are required before the stopping rule is satisfied.

Figure 6 shows three FORTRAN representations of a logically equivalent reasoning process. The three examples of FORTRAN correspond to steps in

the conversion process from the expert system to conventional code. To achieve the direct representation, all clauses would be assigned a variable name and the rules would be translated directly into code. In the second step of the conversion process, all clauses that are only used as antecedents are taken as the basic input symbols (primitives), and then each rule is expanded until it is expressed entirely in terms of these primitives. (Here, to simplify the discussion, these primitives are assumed to represent the input data to the expert system.) The final simplified FORTRAN code is established by applying standard methods for reducing Boolean expressions, such as the Quine-McCluskey minimization method [7]. A set of rules to be used by a forward-chaining inference mechanism is thus transformed into a single-pass, conventional set of higher order language expressions.

Backward-Chaining Example

Backward-chaining inference is often referred to as "goal-driven" inference because the inference mechanism begins with an ordered list of goals (hypotheses) and uses the knowledge base to attempt to find a set of rules that allows these hypotheses to be concluded. If the first hypothesis cannot be satisfied, using the knowledge base and established facts, the second hypothesis is attempted. This process continues until a hypothesis can be asserted or until the list of hypotheses is exhausted. A list of rules and an ordered set of hypotheses are given below.

Rule number	Rule	Hypothesis
1	If **a** *and* **b** *and* **c** then **d**	**d**
2	If **e** *and* **f** then **a**	**b**
3	If **g** *or* **h** then **b**	**a**
4	If **i** *and* **j** then **c**	
5	If **c** *and* **a** then **b**	
6	If **e** *and* **i** then **g**	

To illustrate backward-chaining, the facts **e** and **f** are established initially. Because the hypotheses are ordered, the backward-chaining mechanism first attempts to conclude **d**. This is done by finding a rule with **d** as the consequent — in this case, rule 1 above. The backward-chaining mechanism then compares the established facts and attempts to satisfy the rule antecedents from those facts. If **a**, **b**, and **c** are not established facts (and, in this case, they are not), the backward-chaining mechanism must repeat the process of finding rules with each of the sub-hypotheses as consequents. In doing so, it must test those rule antecedents against the established facts and continue until either the list of rules has been exhausted or all antecedents for some hypothesis can be satisfied. Attempting to assert **d** results in the search-tree shown in Fig. 7. In this example, the hypothesis **d** would be abandoned and the backward-chaining mechanism would test to see if the next hypothesis **b** could be asserted. Given the rules and established facts in this example, **a** can be asserted using rule 2. The backward-chaining mechanism would assert **a**, after trying to assert **d** and **b** in turn, and stop.

Figure 8 shows how the rules and hypotheses discussed above could be represented as FORTRAN code. As in the forward-chaining example, the conversion from rules to FORTRAN code is a three-step process. To achieve the direct representation, all clauses would be assigned variable names and the hypotheses would be translated directly into the code. The second step of the transformation process once again requires that the clauses that are used only as antecedents be identified as primitives (and that these primitives are assumed to represent input data). The representations of the ordered hypotheses are expressed in terms of the primitives. The final step of the transformation again requires the application of a method for reducing Boolean expressions and results in the FORTRAN code presented as the "representation after substitution and reduction" in Fig. 8. The difference between this single-pass code and the backward-chaining example is that *all* hypotheses that can be satisfied will be satisfied. To provide an equivalent mechanism to the backward-chaining example, a test and return must be inserted after each representation of a hypothesis (Fig. 8) and a variable (NULHYP) created to indicate when no hypothesis could be satisfied.

Comparison of Example Expert Systems and Their Conventional Code Representations

In the examples of forward- and backward-chaining, the conversion of rules into a conventional code results in a logically equivalent representation of the knowledge base and inference mechanism. However, the flexibility of the expert systems has been converted into the rigidity of a conventional code. Yet if this conversion is automatic, nothing has been lost. Another step in the development process has been inserted (the conversion process). However, this is a minor inconvenience when considered in the context of a program that would execute faster and could easily be hosted on any of a number of numeric processors. Obviously, the conventional code is less readable and self-documenting than the rules would be for an expert system. Nevertheless, the readability and code documentation are critical only if the code must be maintained and modified by humans. In the processes described above, modification and maintenance of the knowledge base would occur within the context of the expert systems development environment. When the knowledge base is modified, a new piece of conventional source code could be generated and the old code could be discarded. This process would then be similar to the use of a standard compiler. In fact, a *knowledge compiler* is exactly what is being proposed in this paper.

It is important to understand what would be lost in the transformation of expert systems to

conventional code in the examples given. While the conventional code is logically equivalent to the knowledge base of the example expert systems, no provision is made in these examples of a conventional code for an explanation feature. At the user interface, no provision is made for rule modification or even for the display of rules. As discussed in the Avionics Applications of Expert Systems section, the provision for knowledge base maintenance in the application system is neither required nor desirable, but some form of explanation feature is, at times, desirable.

A first-level explanation can easily be generated using a subroutine (procedure) in addition to the subroutine that performs the logical inference. This subroutine would, in essence, contain formatted representations of the knowledge base rules. Returning to the example rules in Figs. 2 and 3 and assuming that all rules would be used in situations requiring explanation, the subroutine shown in Fig. 9 could be generated from the rules to provide the needed explanation. In the example shown in Fig. 9, the explanation in the format statement would be displayed to the user (pilot) whenever the logical variable (C1, C2, C3, C4, or C5) corresponding to the rule consequent of the formatted rule was true. That is, if the equivalent of the first rule in Fig. 2 could have been used to conclude that the "longitudinal rate damping mode is inoperative" and C1 in Fig. 3 would have been true, then the rule encoded in format statement 101 in Fig. 9 would be displayed as an explanation.

Concluding Remarks

A brief introduction to expert systems is presented in this paper. Expert systems are compared to conventional programs and then discussed within the context of avionics applications. After describing the need for expert systems in avionics applications, the problems posed by this technology are discussed. Two example inference mechanisms (forward- and backward-chaining) are described and used to exemplify the proposed technique of converting expert systems knowledge bases into conventional programs.

This paper presents the concept of using a *knowledge compiler* to convert the knowledge base developed for an expert system into a conventional program. This concept allows the most desirable features of expert systems to be retained and also provides a means for producing fast, efficient code capable of execution on any processor. While discussed within the context of avionics applications, this concept has utility in other applications where execution speed is not the primary consideration but where the options available for target machines are limited.

While the expert systems examples presented in this paper are extremely simplified, they are representative of two powerful inference mechanisms that are applicable to a wide range of problems. By demonstrating that the knowledge bases and inference mechanisms used in these example expert systems can be converted into conventional programs, it has been shown how some of the problems of using expert systems in avionics applications may be minimized.

In fact, the proposed approach need not be limited to avionics applications. For any system that can be converted using this technique, the advantages are significant. While the use of symbolic processors in research and development laboratories has many benefits, the costs associated with these single-user, special-purpose systems may make them unsuitable for target applications. The technique described in this paper provides a means for converting expert systems from symbolic processors to numeric processors.

References

[1] J.J. Helly, Jr., "A distributed expert system for space shuttle flight control," Ph.D. Thesis, Univ. of California, Los Angeles, 1984.

[2] V.A. Regenie and E.L. Duke, "Design of an expert-system flight status monitor." NASA TM-86739, Aug. 1985.

[3] E.L. Duke and V.A. Regenie, "Description of an experimental expert system flight status monitor." NASA TM-86791, Oct. 1985.

[4] A. Barr and E.A. Feigenbaum, The Handbook of Artifical Intelligence, vol. I. Los Altos, CA: William Kaufmann, Inc., 1981.

[5] R.J. Brachman, S. Amarel, C. Engleman, R.S. Engelmore, E.A. Feigenbaum and D.E. Wilkins, "What are expert systems?" in Building Expert Systems, F. Hayes-Roth, D.A. Waterman and D.B. Lenat, Ed. Reading, MA: Addison-Wesley, 1983, pp. 31-57.

[6] F. Hayes-Roth, D.A. Waterman and D.B. Lenat, "An overview of expert systems," in Building Expert Systems, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, Ed. Reading, MA: Addison-Wesley, 1983, pp. 3-29.

[7] F.J. Hill and G.R. Peterson, Introduction to Switching Theory and Logical Design, 2nd ed. New York: John Wiley & Sons, Inc., 1974.

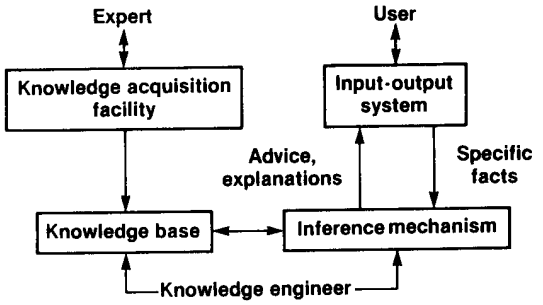

Fig. 1. Basic expert systems architecture.

If
the primary pitch rate gyro has failed
and the backup pitch rate gyro has failed
then
the longitudinal rate damping mode is inoperative

If
the pilot has selected the air-to-air gun mode
or the guidance system has selected the air-to-air gun mode
then
the flight control system is in the fuselage pointing mode
and the air-to-air gun mode indicator is on

If
the mission is interception
and the fuel is not sufficient for a minimum-time interception with maximum thrust
then
use the minimum-time to a cruise energy algorithm

If
the ground-based threat data base has been updated
and the risk associated with the new situation is unacceptably high
then
the route should be replanned

Fig. 2. Example rules for avionics applications.

```
C1  =(A1  .AND.  A2  )
C2  =(A3  .OR.   A4  )
C3  =(A3  .OR.   A4  )
C4  =(A5  .AND.  A6  )
C5  =(A7  .AND.  A8  )
where
A1  =the primary pitch rate gyro has failed
A2  =the backup pitch rate gyro has failed
A3  =the pilot has selected the air-to-air gun mode
A4  =the guidance system has selected the air-to-air
     gun mode
A5  =the mission is interception
A6  =the fuel is not sufficient for a minimum-time
     interception with maximum thrust
A7  =the ground-based threat data base has been
     updated
A8  =the risk associated with the new situation
     is unacceptably high
C1  =the longitudinal rate damping mode is
     inoperative
C2  =the flight control system is in the fuselage
     pointing mode
C3  =the air-to-air gun mode indicator is on
C4  =use the minimum-time to a cruise energy
     algorithm
and C5  =the route should be replanned
```

Fig. 3. FORTRAN representation of example rules for avionics applications.

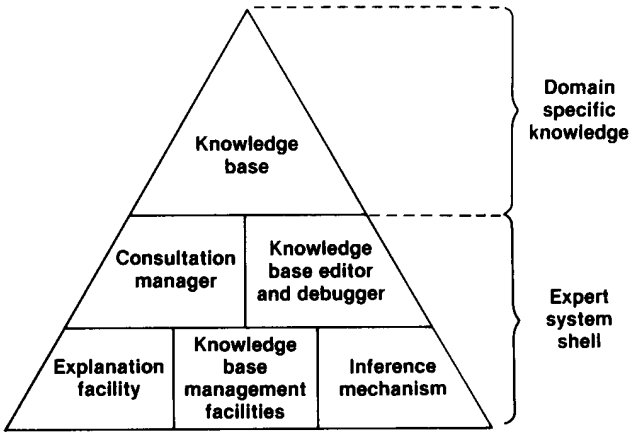

Fig. 4. Components of an expert system.

Inference cycle	Facts established	Rule used
0	(a)	–
1	(ab)	4
2	(abc)	3
3	(abcd)	1
4	(abcde)	2
5	(abcde)	–

Fig. 5. Example of forward-chaining.

Direct representation:

```
D   = C .OR. F
E   = D
C   =(B .AND. A)
B   = A
```

Representation after substitution:

```
D   =((A .AND. A) .OR. F)
E   =((A .AND. A) .OR. F)
C   = (A .AND. A)
B   =  A
```

Representation after substitution and reduction:

```
D   =A .OR. F
E   =A .OR. F
C   =A
B   =A
```

Fig. 6. FORTRAN representation of forward-chaining example.

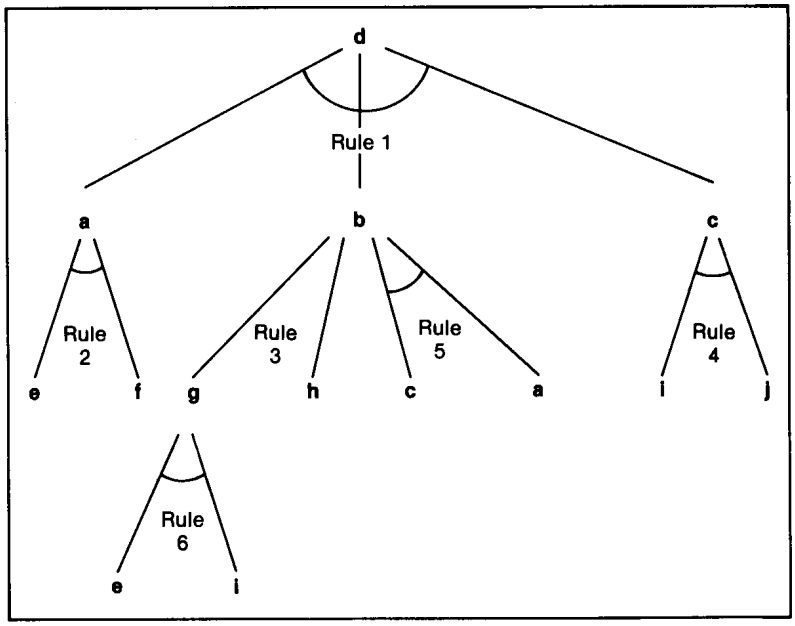

Fig. 7. Search-tree associated with example hypothesis.

```
Direct representation:
  D  =((A .AND.  B) .AND.     C)
  B  =((G .OR.   H) .OR.   (C  .AND.   A))
  A  = (E .AND.  F)

Representation after substitution:
  D  =(((E .AND. F) .AND. ((E .OR. I) .OR. H)) .AND. (I AND. J))
  B  =(((E .AND. I ) .OR. H) .OR. ((I .AND. J) .AND. (E .AND. F)))
  A  =  (E .AND. F)

Representation after substitution and reduction:
  D  =(((E .AND. F) .AND. I) .AND. J)
  IF D RETURN
  B  = ((E .AND. I) .OR. H)
  IF B RETURN
  A  =  (E .AND. F)
  IF A RETURN
  NULHYP=. TRUE.
```

Fig. 8. FORTRAN representation of backward-chaining example.

```
      SUBROUTINE EXPLAN
C
      COMMON /CFACTS/ C1 ,C2 ,C3 ,C4 ,C5
C
      IF (.NOT. EXPLIN) RETURN
      IF (C1 ) WRITE (*,101)
      IF (C2 ) WRITE (*,102)
      IF (C3 ) WRITE (*,103)
      IF (C4 ) WRITE (*,104)
      IF (C5 ) WRITE (*,105)
      RETURN
C
101   FORMAT(" The longitudinal rate damping mode is",
     .       " inoperative",/,
     .       " because the primary pitch rate gyro has failed",/,
     .       " and the backup pitch rate gyro has failed.")
102   FORMAT(" The flight control system is in the fuselage",
     .       " pointing mode",/,
     .       " because either the pilot has selected the",
     .       " air-to-air gun mode",/,
     .       " or the guidance system has selected the",
     .       " air-to-air gun mode.")
103   FORMAT(" The air-to-air gun mode indicator is on",/,
     .       " because either the pilot has selected the",
     .       " air-to-air gun mode",/,
     .       " or the guidance system has selected the",
     .       " air-to-air gun mode.")
104   FORMAT(" Use the minimum-time to a cruise energy",
     .       " algorithm",/,
     .       " because the mission is interception",/,
     .       " and the fuel is not sufficient for a",
     .       " minimum-time interception with maximum thrust.")
105   FORMAT(" The route should be replanned",/,
     .       " because the ground-based threat data base has",
     .       " been updated",/,
     .       " and the risk associated with the new situations",
     .       " is unacceptably high.")
      END
```

Fig. 9. FORTRAN subroutine to provide first-level explanation.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-88263		

4. Title and Subtitle	5. Report Date
AN ENGINEERING APPROACH TO THE USE OF EXPERT SYSTEMS TECHNOLOGY IN AVIONICS APPLICATIONS	May 1986
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Eugene L. Duke, Victoria A. Regenie, and Marylouise Brazee (NASA Ames-Dryden), and Randal W. Brumbaugh (PRC Kentron)	H-1364
	10. Work Unit No. RTOP 505-66-11

9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Ames Research Center Dryden Flight Research Facility P.O. Box 273 Edwards, CA 93523-5000	
	13. Type of Report and Period Covered: Technical Memorandum

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15. Supplementary Notes

Prepared as an invited IEEE paper for presentation at National Aerospace and Electronics Conference (NAECON), Dayton, Ohio, May 19-23, 1986.

16. Abstract

This paper presents the concept of using a *knowledge compiler* to transform the knowledge base and inference mechanism of an expert system into a conventional program. The motivation for this discussion is the need to accommodate real-time systems requirements in applications such as embedded avionics. The paper presents an over-view of expert systems and a brief comparison of expert systems and conventional programs. Avionics applications of expert systems are briefly discussed before the detailed discussions of applying the proposed concept to example systems using forward- and backward-chaining.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Avionics Expert systems Knowledge compiler	Unclassified – Unlimited STAR category 05

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	10	A02

**For sale by the National Technical Information Service, Springfield, Virginia 22161.*